Scalable Cluster Administration - Chiba City I Approach and Lessons Learned

John-Paul Navarro, Rémy Evard, Dan Nurmi, Narayan Desai
Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, IL 60439, USA
{navarro, evard, nurmi, desai}@mcs.anl.gov

Abstract

Systems administrators of large clusters often need to perform the same administrative activity hundreds or thousands of times. Often such activities are time-consuming, especially the tasks of installing and maintaining software. By combining network services such as DHCP, TFTP, FTP, HTTP, and NFS with remote hardware control, cluster administrators can automate all administrative tasks.

Scalable cluster administration addresses the following challenge: What systems design techniques can cluster builders use to automate cluster administration on very large clusters? We describe the approach used in the Mathematics and Computer Science Division of Argonne National Laboratory on Chiba City I, a 314-node Linux cluster; and we analyze the scalability, flexibility, and reliability benefits and limitations from that approach.

Introduction

In 1994 Thomas Sterling and Don Becker, then working for NASA, built the first commodity PC-based cluster using sixteen 486 DX4 systems. Commodity PC-based clusters quickly gained momentum, and today many educational and research organizations choose them as the high-performance computing platform of choice

The first step in automating cluster administration generally involves setting up one or more hosts to provide services used in automated boot, build, and monitor and to perform other cluster administration activities. Common administration support services include

- DHCP for network initialization
- TFTP for boot image delivery
- FTP, HTTP, NFS, and SMB for script and software package delivery
- A configuration information delivery service (database or file based)
- SNMP for monitoring and event notification
- SYSLOG for OS and core service logging
- DNS for host name resolution
- NTP for time synchronization

In this paper we call machines providing these services “management servers”. Most clusters require very few management servers. For all small clusters, and most medium to large clusters, an organization’s existing infrastructure servers may be capable of providing these services to clusters.

To decide whether one or a small set of management servers is sufficient, one must determine how many clients will concurrently use management services and how quickly these management service requests must be handled.

Services used during different phases in a machine’s life cycle may have totally different

usage patterns. Network interface configurations with DHCP and boot image delivery with TFTP are generally only used at machine boot time. File delivery using FTP, HTTP, NFS, and SMB are used heavily during machine build or upgrade. Logging (SYSLOG), monitoring (SNMP), and remote console services are most important during normal operations and typically have constant, low-level requirements.

As the size of a cluster grows and the build and reconfiguration rate increases, the demand for these services may exceed the capabilities of a single machine or of the available infrastructure servers. Possible first steps to address this management scalability limitation include separating services onto different machines and upgrading servers so they can service more requests. These steps are generally enough to scale most management services for large clusters.

What exactly are the scalability characteristics of the various services? Some services, like file delivery services, may need to deliver hundreds of megabytes in order to build a single node. How does one scale these services so that hundreds of nodes can rebuild in a short period of time? This paper present the scalable cluster management approach used to address these and similar questions on Chiba City I at Argonne National Laboratory.

Importance of Management Scalability

For most clusters, management scalability is not an issue, because the number of managed nodes is low, the rebuild and reconfigure rate is low, or rebuild and reconfiguration performance is not a concern.

Management scalability matters on Chiba City because its primary purpose is to be a ***scalability testbed***, built from ***open source*** components, for the high-performance computing and computer science communities. As a scalability testbed, Chiba City is dedicated to the research, development, and testing of architectures, algorithms, software, and protocols that push the scalability boundary of clusters and the applications that run on them.

Although built and operated primarily using open source software, the goal is for Chiba City software to support installation and operation of any open or closed-source operating system on the non-management nodes. In support of this objective we developed a cluster administration toolkit, called the City Toolkit, designed to support the unattended installation of arbitrary operating systems. Because Chiba City is a testbed, we need to rebuild and reconfigure nodes frequently and as quickly as possible. The combination of a relatively large number of nodes and a high rebuild and reconfiguration rate makes a scalable cluster management design critical.

With a scalable cluster management design rebuilds, reconfigurations, and general management activities can run quickly improving efficiency and overall cluster availability.

Today, using the City Toolkit, we can change a master configuration database that specifies the desired image (OS + configuration) for managed nodes and complete a parallel rebuild of all nodes in under 30 minutes. Our long-term goal is to support node rebuilds on demand, for example, to support rebuilding nodes based on OS requirements specified in a batch job, or to be able to quickly assign 500 nodes with a custom kernel to a kernel developer for scalability testing and debugging.

Another reason scalable management matters is that with the widespread adoption of clusters for high-performance computing, we believe

the research and commercial communities will deploy clusters with tens and hundreds of thousands of nodes in the next 20 years. As a scalability testbed, Chiba City must be able to provide and environment where researchers can investigate scalable management challenges that will undoubtedly affect all very large commodity PC-based clusters.

Chiba City Management Architecture

Figure 1 shows all the components in Chiba City.

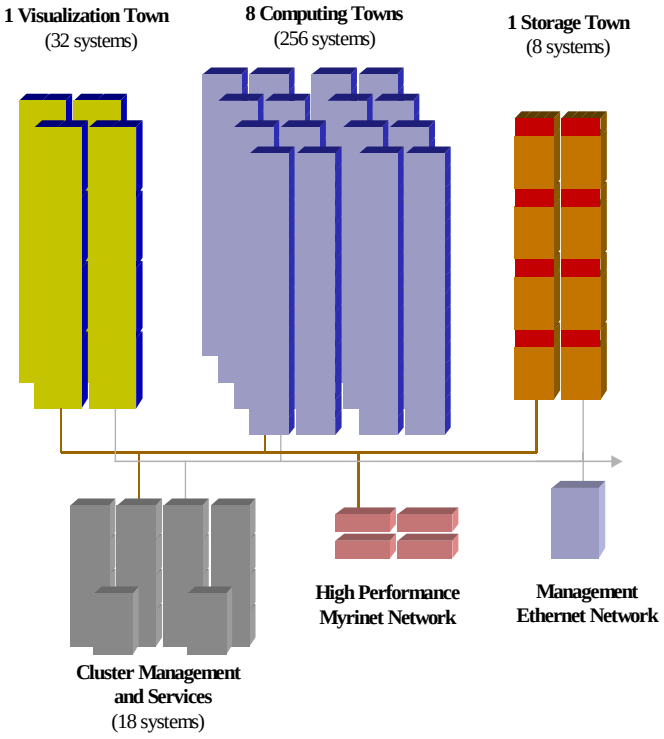


Fig. 1. Diagram of Chiba City components

Physically, Chiba City comprises blocks of physically contiguous nodes called towns. A town consists of between 8 and 32 application-usable nodes, one management node called a mayor, an Ethernet network switch, serial concentrators connecting node serial ports to the mayor, and network-addressable remote power controllers. Most Chiba City towns fit into two racks. The only management-related cables coming from the two town racks are dual Gigabit Ethernet uplinks, one power controller Ethernet network uplink, the mayor's serial port cable, and five power cables. A town has all the hardware necessary to operate as an independent subcluster. As a matter of fact we've taken one such Chiba City town to a conference.

Towns are not merely a physical partitioning scheme. The concept defines how management software and services--required to build, configure, and operate the nodes in a town--work. An example of that physical and management link is the boot-and-build process for nodes. When a node boots, it sends its boot loader prompt (LILO or GRUB) over its serial port. The town mayor, which runs serial port monitoring software, detects that boot loader prompt and sends a response, telling the node either to boot or build. The build tools used to build a node and the software packages installed on that node are accessed from the mayor via NFS or HTTP/FTP.

The mayor provides the following services:

- Relay or proxy access to the master cluster database
- DHCP for configuring network interfaces
- Tftp access to boot images
- Console monitoring, logging, and interactive access
- Rootnfs used during node build and for node debugging
- NFS access to shared user software
- NFS, HTTP, or FTP access to software packages installed during a build
- NFS-based file relay service for copying files to or from the nodes before and after job runs (Chiba City doesn't have a global general-purpose file system)

- Relay service for global management commands

Managing the town mayors, login machines, file servers, and a batch scheduler server is a master management server called the president. The president behaves exactly like a mayor to the nodes it manages. The most significant difference between mayors and the president is that the president contains the master copy of the images (OS + configuration), layered software packages, and the master cluster configuration database. Software is synchronized between the president and the mayors using rsync. The master configuration database on the president is accessed using proxies on mayors.

We chose this three level hierarchical design because we felt it could architecturally be extended to more levels, thus providing a form of scalability. Using a ratio of32 managed nodes to one management server we could build a 1024-node cluster with the current three levels in Chiba City. Using the same ratio but expanding the design to four hierarchical levels, we could operate a 32,000 node cluster.

As an example of how this management hierarchy affects administrative activities, the following steps describe the process of modifying a node image and rebuilding a node using the modified image. Briefly, images as we defined them in the City tools are the combination of disk initialization information (partitions, file-system types, and initial file-system contents), a collection of scripts that drive the build process, and a set of software packages and configuration files.

1. Change the appropriate node image on the president (for example, change the desired root file-system size).
2. Rsync all the images from the president to the mayors.
3. Modify the master configuration database, specifying which nodes need the new image.
4. Reboot or remotely power cycle the nodes that need the new image.

The nodes boot and are directed by their mayor to build based on the database specified image. When the nodes finish building they record so in the master configuration database

Elements of the Chiba City Management Architecture That Scaled Well

Overall we found many aspects of the Chiba management architecture worked well on a 314-node cluster. The following sections describe these aspects that, in principle, we feel could be used to manage clusters with tens of thousands of nodes.

Dedicated Management Servers

An important aspect of our design is that management servers are not available to user applications. Although that design decision seems obvious to us, many people have confronted us about the added hardware tax. The separation is particularly important in our environment because most cluster applications expect dedicated access to nodes. Our cluster build-and-configure software was designed to run on Linux machines. The software requirements for the management services are often different from those of the application software. For example, currently our management and software runs under RedHat 7.1 but is capable of installing other versions of RedHat or even totally different distributions such as Mandrake and operating systems such as FreeBSD.

Specialized Management Server Hardware

Management servers need a hardware configuration targeted to running management service. This hardware configuration often differs from what user application nodes need. For example, management servers often require hundreds of gigabytes of disk to store or cache software packages, high-performance gigabit commodity networks to handle high conventional TCP/IP communication loads, and multiple CPUs to handle very high context switching rates associated with many clients making management service requests. To fulfill these hardware requirements, one generally must use high-end large footprint hardware for management servers. In contrast, large clusters typically look for very small footprint machines to be the workhorses used by cluster applications.

Master Management Server

Once settled on dedicated management servers, one has to determine how to distribute services between those servers. The approach used on Chiba City was to give a single machine the role of master management server which we called the president. A president was the authoritative source for all software and configuration information, the recipient of all status information, and the point from which all administrative function could be issues. We found this model to be straightforward and easy to use. From the president we had the ability to build, configure, and update other management servers and to initiate management operation that would be forwarded to the mayors responsible for the desired target nodes. We believe that a canonical source for all software, from which all management commands might be issued, should scale to any size cluster. For this to be the case, though, management software must divided and delegated to operations to subordinate management servers.

Using the master management server as the home for the master configuration and state database worked well from a conceptual point of view. We have database performance issues that will be discussed below.

Rebuildable Management Nodes

Once we established that we had single master management server, the president, we were able to build procedures for automatic rebuilding of all other management servers. It is even possible to rebuild a management node and then transfer the master president role to it, in effect upgrading the president. The scalability advantage here is that by automating management server builds, it becomes easy to add more management servers.

Remote Power Control

Remote network-based power control is an essential component of hands-off administration. Without it, some administrative and operational activities would require individuals walking up and down aisles of computer racks pushing buttons. This is impractical, not just from a human time/cost perspective, but also because of the likelihood of pushing the wrong button.

Remote Console

We found a remote console to be a useful tool for node identification, boot control, and network failure diagnosis and recovery. In vary large clusters a remote console may not be essential if network or other techniques are used to identify nodes and control the boot process.

Parallel Management Algorithms

Early in the City Toolkit design we realized that along with parallel hardware and management servers independently managing a

subset of nodes, we also needed parallel management software algorithms. One very effective tool we relied on heavily is pdsh (`http://www.llnl.gov/icc/lc/pdsh.html`). Pdsh provides a simple thread based model for executing commands on multiple nodes in parallel.

As mentioned above, another essential design point in scalable management software is delegation. By taking a single centrally issued command, splitting it into parallel components, and delegating those to other management servers arbitrary management operations can scale to thousands of nodes and execute quickly.

Elements of the Chiba Management Architecture That Did Not Scale Well

Intro

Single point of failure

The master management server (i.e. president) is a single point of failure. When this server fails it may be impossible to issue administrative commands or to perform any operations that depend on the master configuration services.

Management server hardware configuration

Specking and selecting the appropriate management server hardware is more important than we had originally thought. Care should be taken that the appropriate amount of RAM, CPUs, disk space, I/O bandwidth, and network bandwidth is available on management servers.

Hard mapping between managed and management nodes

The most unscalable and problematic aspect of the Chiba management architecture was the direct dependence between a fixed set of contiguous nodes and a single management server. One consequence was very poor management service load distribution which caused poor service response due to overloading on some management machines others were effectively idle.

One example of how this might happen is that the high performance cluster interconnect has better performance between nodes that are physically close together. To take advantage of this the scheduler needs to assign nodes to jobs in a contiguous range. Rebuilding or reconfiguring nodes in a contiguous range would place all the load on one or a few set of management nodes while the rest are idle.

Management node failures affect a large number of nodes.

When a mayor fails, all of the nodes beneath it in the hierarchy cease to operate. It's also impossible to rebuild or reconfigure them because the client to mayor link in wired in hardware and software designs.

Console access through management nodes

Console access tied to mayors meant that if a particular mayor was unavailable all the nodes it managed could be unusable if the depended on a management service like nfs based software. Also we could not rebuild or reconfigure them since management services could only be provided by that one management server.

All management servers providing most services

Putting all management services on every mayor created a headache for protocols that could easily be serviced by a single machine.

For example DHCP and the configuration database. In our particular case where all Chiba City was under a single flat IP space, running 11 DHCP servers on the same subnet became a challenge.

Some services aren't suited for a hierarchy

Some management services are difficult to install and operate hierarchically (and don't require or benefit from it).

Picking the right management server to managed node ratio for services

It's impossible to pick the "right" ratio of mayor to managed nodes since each service has different scaling characteristics. For example, the number of concurrent NTP or DHCP requests that a single server can handle is probably in the 100s or 1000s, while the number of HTTP or FTP get requests for a large RPM package is probably more in the 10-20 range (if the management node has approximate 10x better Ethernet bandwidth than the nodes it manages). If both of these classes of services are configured in a fixed 32 client to 1 server ratio we effectively using more NTP and DHCP servers than we need and fewer HTTP and FTP servers than could be effectively utilized.

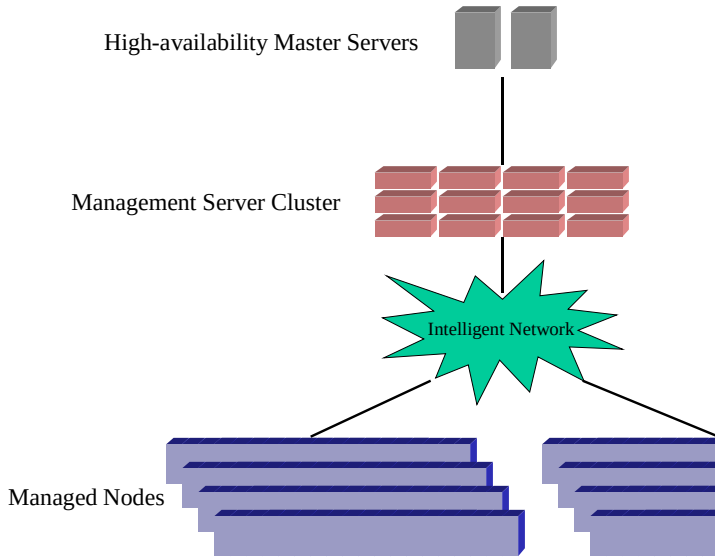
Hierarchical configuration and software push

Pushing configuration changes and commands down the 3 level hierarchy is a time consuming process. Having to deal with 4 or 5 levels would be a serious problem. For example, to apply software or configuration changes to nodes requires applying the change on the president, pushing the change iteratively down the hierarchy, and finally apply the change on the target machines. This multi-step, fixed path, design is very vulnerable to cascading failures. At one point we had 200 MB of software required on every management node, pushing this much software every time we changed something was a slow and painful process, even when using an intelligent push tool like rsync.

Future work

The Chiba City scalable management approach has taught us several lessons both about what works well and what doesn't. Moving forward we've identified a new management architecture we intend to investigate that could be used to manage 1000s of nodes.

To avoid the problems associated with a multi-level hierarchy while taking advantages of the master and authoritative configuration and state server we are beginning to explore a fixed 3 layer architecture.



The top level is the master and authoritative configuration and state server that we call the president. The middle tier contains a collection of management servers and devices that are

directly responsible for providing the network services used to build the non-management nodes. The final tier contains all the managed nodes used by users and applications.

At first glance this architecture appears very similar to the current Chiba City I architecture. It's different in the following ways: first, there are only three levels in any size cluster, and second, the middle tier can itself be thought of as a cluster of machines dedicated to running management services/applications. In this tier there may be many different server configurations dedicated to specific management functions. There would be not hard tie between managed nodes and specific management nodes. Using either intelligent network hardware or dynamically reconfigurable protocols between managed and management nodes. We also see the possibility of using parallel programming techniques, like MPI, to write parallel management services. The number and type of nodes used for each type of management service would be flexible and based on individual protocol requirements. Some services may require a single machine for almost any size cluster, for example, NTP or DHCP. While other services, like FTP and HTTP may require multiple machines behaving as a load balanced mini-cluster.

We also intend to investigate the use of high-availability techniques at the master server/president level to reduce the likelihood of not being able to operate the cluster when the single most critical component in the cluster fails.

We are investigating using automated caching techniques in the middle tier to avoid pushing updates from the master management server down.

We've also identified two interesting approaches to load balancing and mapping managed nodes to management nodes. First the use of intelligent networking hardware and second more client centric techniques like client selected round robin with failure detection and retry.

We see parallelization of management application and command as essential technique for scalable cluster management.

Reference:

www.beowulf.org Information on Beowulf and Commodity PC based clusters.

D.J. Becker, T.L. Sterling, D.F. Savarese, J.E. Dorband, U.A. Ranawak, and C.V. Packer. Beowulf: A Parallel Workstation for Scientific Computation. *Proceedings of the International Conference on Parallel Processing*, 1995.

T. Sterling, D. Becker. How to Build a Beowulf.

www.top500.org. Top 500 super-computers in the world.

D. S. Greenberg, R. B. Brightwell, L. A. Fisk, A. B. Maccabe, and R. E. Riesen. A System Software Architecture for High-End Computing. *Proceedings of Supercomputing '97*, 1997.

J. Challenger, P. Dantzin, A. Iyengar. A Scalable and Highly Available System for Serving Dynamic Data at Frequently Accessed Web Sites. *Proceedings of Supercomputing '98, 1998.*

B. Dempsey, D. Weiss. On the Performance and Scalability of a Data Mirroring Approach for I2-DSI. *Network Storage Symposium Proceedings, 1999.*

The City Toolkit: http://www.mcs.anl.gov/systems/software/

The parallel distributed shell (PDSH): http://www.llnl.gov/icc/lc/pdsh.html

J. Challenger, A. Yyengar, P. Dantzig. A scalable system for consistently caching dynamic web data. Proceedings of IEEE INFOCOM ’99, p. 294-303. March 1999, New York.

A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, P. Gauthier. Cluster-based scalable network services. Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles. P. 78-91. October 1997, Saint Malo, France.

B. Gronvall, A. Westernlund, S. Pink. The design of a multicast-based distributed file system. Proceeding of the Third Symposium on Operating Systems Design and Implementation. P. 251-64, February 1999, New Orleans.

V. Jacobson. Congestion avoidance and control. SIGCOMM ’88 Symposium: Communications Architectures and Protocols., p. 314-29, August 1988, Stanford.

A. Luotonen, K. Altis. World-Wide Web proxies. Computer Networks and ISDN Systems 27, no. 2: p. 147-54, November 1994.

V. S. Pai, M. Aron, G. Banga, M. Svendsen, P. Druschel, W. Zwaenepoel, E. Nahum. Locality aware request distribution in cluster-based network servers. ASPLOSS-VIII. Eighth International Conference on Architectural Support for Programming Languages and Operating Systems, p. 205-16, October 1998, San Jose.